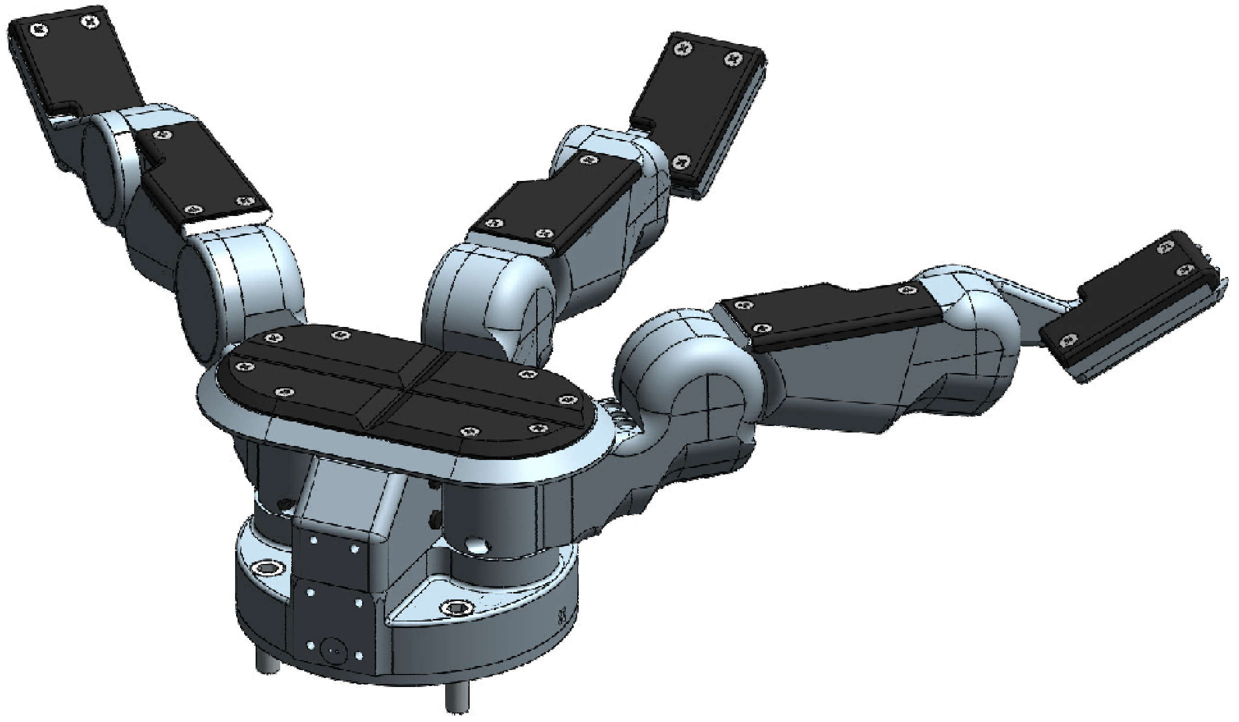
BarrettWraptor™
BH8-600 Series User Manual
Firmware Version 1.0



Barrett Technology Inc.

1 System Description

1.1 Introduction

Thank you for choosing the BarrettWraptor™. The BarrettWraptor ™ is designed to overcome the inflexibility of conventional industrial grippers with DSP-enabled dexterity while maintaining durability, compactness, and ease of use. The BarrettWraptor ™ is a multi-fingered Wraptor™ with the dexterity to secure target objects of different sizes, shapes, and orientations. Rather than rely on pinching gripper friction or permanent gripper-jaw shape customization the BarrettWraptor ™ gently envelops the object, securely locking its joints until commanded to release.

System integration with any robotic arm is fast and simple. Even with its low, 6.9-kg, weight and compact form, it is totally self-contained. The BarrettWraptor ™ uses industry-standard serial communications, which is the common denominator of communications, for guaranteed universal compatibility. CAN communications and Ethernet are available, optionally. While the bandwidth of serial communications may limit less intelligent equipment, the eight (8) on-board DSPs combined with Barrett's open Grasper Control Language (GCL) endows the BarrettWraptor ™ with millisecond response.

The compactness and low weight of the BarrettWraptor ™ assures that the enhanced dexterity does not compromise arm payload. Its low mass and short base-to-grasp-center distance minimize joint loading on the host robot and reduce extraneous arm movements during object reorientation. The custom control-electronics package is contained entirely within the Wraptor™, reducing electrical wiring to a single cable carrying all communications and motor power.

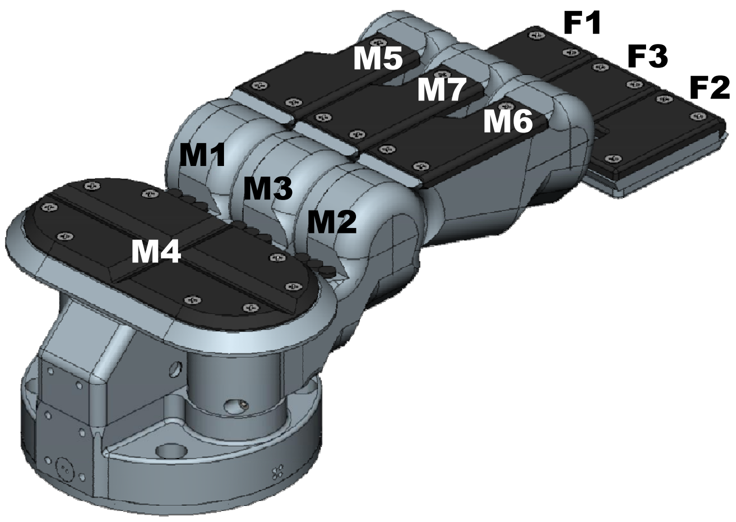
We hope that you enjoy the versatility and functionality of the BarrettWraptor ™. Please never hesitate to give feedback and to ask for advice as needed. US+617-252-9000, <service@barrett.com>, or <http://www.barrett.com/robot/>.

1.2 About the BarrettWraptor

The BarrettWraptor ™ has three fingers identified as F1, F2 and F3. Two of the fingers, F1 & F2, rotate synchronously and symmetrically about the base joint in a spreading action. The "spread" motion around the palm allows "on-the-fly" grasp reconfiguration to adapt to varying target object sizes, shapes, and orientations.

Aside from the spread motion, each of the three fingers on the BarrettWraptor ™ feature two joints driven by separate DC brushless servo motors. When the inner joint contacts an object, the outer joint continues to close around it, making a secure grasp. Using the fingers together allows the BarrettWraptor ™ to "grasp" a wide variety of objects securely. The multi-jointed fingers, combined with the spread function, make object grasping nearly target-independent.

Two sets of four LEDs are mounted in the base of the Wraptor. The blue LED indicates 24V power. The green LED indicates 3.3V logic power. The yellow LED is lit when the spread motion is free. When the yellow LED is dark, the spread is locked by the brake spread brake. The red LED is a software LED, but it is inactive with the Ethernet model of the Wraptor.



1.3 Technical Specifications

1.3.1 Overview

Dimensions:	131h x 640w x 192d (mm)
Mass:	6.9kg
Voltage requirements:	24V
Current requirements:	Min 10A, Max 30A
Load limits:	50kg / finger
Speed limits (joint extremes):	Outer link = 1.0s
	Inner link = 0.72s
	Spread = 0.88s
Operating temperature limits:	0 to 70 degrees C
Storage temperature limits:	-25 to 95 degrees C
Environmentally sealed:	IP-65

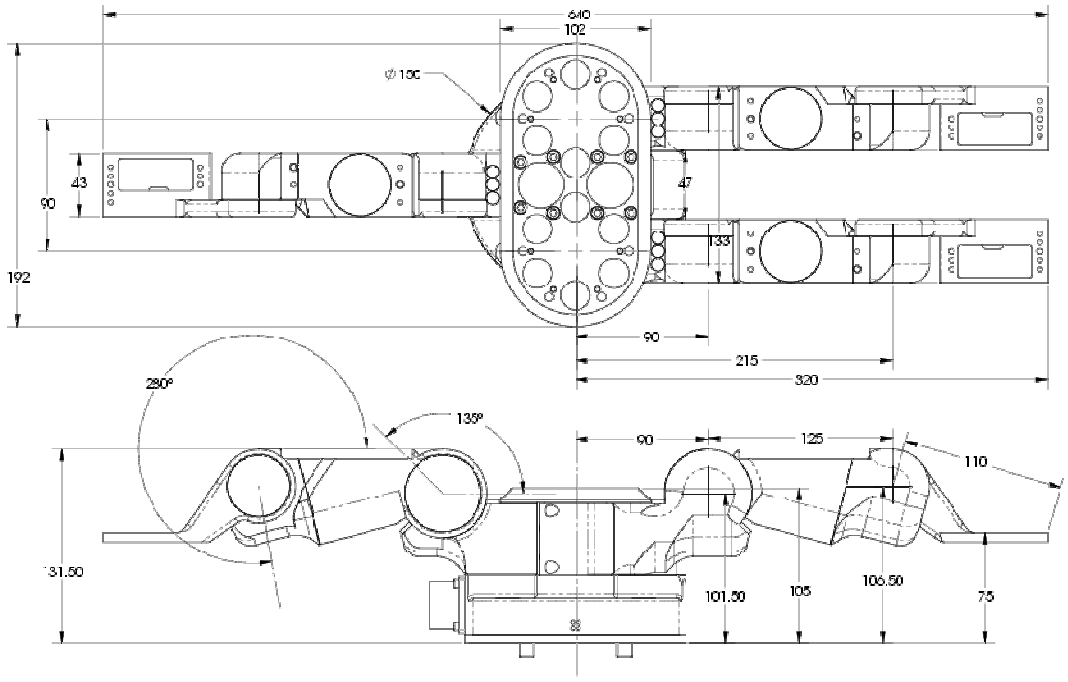


Table 1 - Joint Ranges

		Link	**Spread**	**Tip**
GCL Counts	**Min**	-1375	0	-8690
	Max	18590	9130	19250
Hall Counts	**Min**	-25	0	-158
	Max	338	166	350
Degrees	**Min**	-10	0	-78
	Max	133	180	180

1.3.2 Conversion ratios

The Wraptor operates using hall counts much more imprecise than the GCL specification. Therefore, the physical counts have been scaled to reflect values similar to those in the GCL. When using the GCL, all commands are in the “GCL

Counts" unit. The following is the conversion ratios needed to convert between units and the joint ranges of each motor in all of the different units.

Table 2 – Conversion Ratios

Ratio	Value
GCL Count : Hall Count	55:1
Hall Count : Motor Revolution	18:1
Motor Revolution : Link Revolution	49:1
Motor Revolution : Spread Revolution	18.28:1
Motor Revolution : Tip Revolution	39:1

Example Calculation (Determine the GCL value needed to move link by 90º):

$$90^{o}\left(\frac{1\,link\,\mathrm{Re}\,v}{360^{o}}\right)\left(\frac{49\,motor\,\mathrm{Re}\,v}{1\,link\,\mathrm{Re}\,v}\right)\left(\frac{18\,hallCount}{1\,motor\,\mathrm{Re}\,v}\right)\left(\frac{55\;GCLCount}{1\,hallCount}\right)=12128\;GCLCounts$$

1.3.3 Brushless Motors

The BarrettWraptor ™ utilizes one of the smallest DC brushless servo motors in the world for their torque range. Because the motors have no brushes, and thus less inherent friction, they achieve a better torque/mass ratio than typical brushed servos. There is also no need to replace worn brushes after the motors have been in service over a period of time. The following table shows BarrettWraptor ™ motor properties.

BarrettWraptor™ Motor Properties

Number of Phases	3
Number of Poles	6
Rotor Magnets	Highest-Grade Neodymium Rare-Earth
Commutation	6-step Brushless Electronic PWM
Peak Torque	65 N-cm (92 oz-in)
Motor Constant	4.2 N-$\frac{cm}{\sqrt{W}}$ (5.916 $\frac{oz-in}{\sqrt{W}}$)
Position Feedback	18 counts/rev., Hall effect sensor

1.4 Theory of Operation (Ethernet Model)

The host PC connects and communicates to the Wraptor using Ethernet. A device server inside the Wraptor converts command packets between Ethernet and RS-232 serial (at 38kbaud) to communicate with the GOD (General Operations Device) controller in the base of the Wraptor. If the GOD controller receives an ASCII string command, it interprets the command, compiles it into a valid CAN packet, then places it on the internal CAN (Controller Area Network) bus to be delivered to the motor controllers (Taters). If GOD receives a pre-compiled command packet, it simply places it on the CAN bus.

When GOD receives a CAN packet from a Tater which is addressed to the host PC, it sends the data to the device server via RS-232 serial. If GOD's communication reply mode is set to zero, it translates the data into human-readable text. If it is set to one, it keeps the data in packet-form. The device server then routes the data back to the host PC via Ethernet.

1.5 Control Software

The BH8-600 Series System control software consists of:
1. Firmware (*.tek software), and
2. Example programs.

Included with the software in electronic form are:
1. BH8-600 Series User Manual (this manual)

The BarrettWraptor ™ has firmware that resides on the control electronics inside the palm. Depending on the configuration purchased, you can control the Wraptor™ in one of three ways:
1. RS-232 Serial Communication: Simple, ASCII-based commands are sent over the serial port and interpreted by the Wraptor™, or faster, pre-compiled commands are sent to the Wraptor™ using the C-Function Library.
2. Ethernet Communication: Simple, ASCII-based commands are sent over an Ethernet network and interpreted by the Wraptor™, or faster, pre-compiled commands are sent to the Wraptor™ using the C-Function Library.

CAN Communication: High-speed, pre-compiled commands are sent directly to the motors of the Wraptor™, eliminating the need for an interpreter and allowing absolute control of each motor by using the C-Function Library.

1.6 C-Function Library

The BarrettWraptor ™ C-Function Library is a helpful tool for programming the BarrettWraptor ™ using the C language on IBM-compatible PC's without having to manage the issues of communication and timing of multi-axis motion control. The library contains easy-to-use functions that streamline the development of custom motion control routines by the end user. All of the functions are available when the library is linked to the program.

The C-Function Library is written in C and compiled for Windows 95/98/NT/2k. The library uses a multithreaded mechanism for communications, allowing commands to be issued and responses received simultaneously. The library also manages all input and output buffers and makes it easy to manipulate the BarrettWraptor ™ from a custom control application.

1.7 Control Software/Firmware Upgrades

Barrett Technology makes software and firmware upgrades periodically. Upgrades are available for purchase or free of charge for customers of Barrett's subscription service. Refer to Barrett's enclosed Warranty and Subscription Service Policy for more information.

2 Safety and Cautions

PLEASE READ THIS SECTION IN ITS ENTIRETY BEFORE USING YOUR BARRETTWRAPTOR™.

Following these safety instructions will help prevent user injury and equipment damage.

- As with any piece of robotic equipment, it is ultimately up to you to be aware of your surroundings during robot operation. The workspace of the system comprising the BarrettWraptor ™ and robot arm should be clearly marked to prevent persons or objects from inadvertently entering the equipment's reach. Before attaching the BarrettWraptor ™, test host robot trajectories to confirm that it will not inadvertently collide with other objects in the workspace.

- NEVER connect or disconnect any electrical cables while the Power Supply is turned on. Failure to follow this instruction could impart irreparable damage to the onboard electronics or put you at risk of electrical shock.

- Always make sure the BarrettWraptor ™ is properly grounded. Failure to do so could damage the BarrettWraptor ™ electronics and put you at risk of electrical shock.

- Do not place any part of your body or delicate objects within the grasp of the BarrettWraptor ™ without first verifying control of the unit and confirming appropriate force levels.

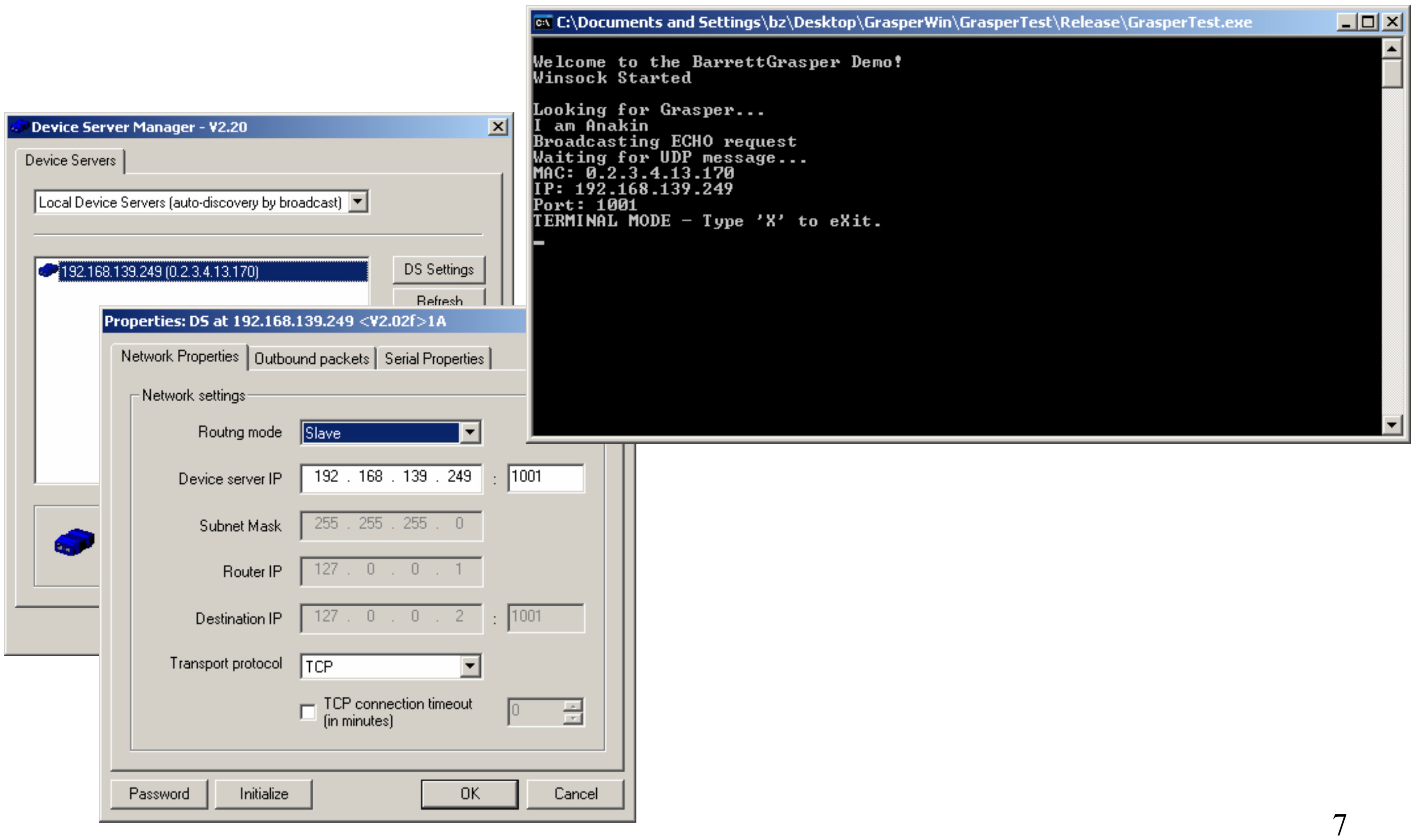
- Do not allow the BarrettWraptor ™ to be exposed to corrosive liquids that may cause damage to the body or cable of the unit.

- Do not exceed the load limit of the fingers, 50kg per finger. Consider all loading situations including accelerated loads, cantilever loads from long objects, robot collisions, active loads, etc.

- Monitor the operating temperature of the BarrettWraptor ™ so that it does not exceed 70C. Under normal conditions, the Wraptor™ operates between 35 and 60C. The BarrettWraptor ™ was designed with non-backdrivable finger joints to take advantage of the motors' peak operating performance in short bursts. The spread, however, is backdrivable to aid in target-independent grasping and requires constant motor current to actively hold position. Idling the spread motor (thus activating the spread brake), when possible, will help keep the temperature lower.

3 Initial Setup and Walk-through

1) Mount the Wraptor securely to a robot arm or test stand.
2) Supply the Wraptor with 24V (red wires) and Gnd (black wires).
3) Plug the Wraptor into an Ethernet network.
4) Launch the "DS Manager" utility, select "DS Settings".
5) Enter a free IP address on your Ethernet subnet. Save your changes and exit the DS Manager.
6) Launch the GrasperTest application.
7) Type "WAKE" to wake up the Wraptor.
   Note: All commands must be followed by pressing <Enter>.
   Note: All commands are case-insensitive and all whitespace is optional.
8) Get the initial temperature of the inner joint motors: Type "123 G TEMP"
   Note: "123 G TEMP" is read as "Motors **1, 2,** and **3, G**et your **TEMP**erature"
   Note: The Wraptor should return with its inner joint temperatures, in tenths of a degree C.
10) Move the fingers to position 100: Type "M", Type "123", Type "100" (three separate entries, <Enter> after each)
11) Move the spread (motor 4) to its mid-position: Type "4 S E 82", Type "4 S M 5"
    Note: "4 S E 82" is read as "Motor **4**, **S**et your **E**nd position to **82**"
    Note: "4 S M 5" is read as "Motor **4**, **S**et your Control **M**ode to **5**"
    Note: The Control Modes are described in detail later, but 0 = Idle, 3 = PID, 5 = Trapezoidal Move. You may notice the Wraptor responding with "x MODE=3" messages (where 'x' is the motor controller number). When a motor completes a trapezoidal move, it signals the host with the information that it has automatically switched its own control mode from trapezoidal (mode 5) to PID (mode 3), and is now holding its end position.
12) Close finger 1: Type "C", Type "1"
13) Open finger 1: Type "O", Type "1"
14) Close the spread: Type "4 S E 164", Type "4 S M 5"
15) Close the Wraptor fingers: Type "C", Type "123"
16) Get the final temperature of the inner joint motors: Type "123 G TEMP"
17) Save the present positions for shutting down: Type "SAVE"
18) Remove power from the Wraptor.


C:\Documents and Settings\bz\Desktop\GrasperWin\GrasperTest\Release\GrasperTest.exe
Device Server Manager - V2.20
Device Servers
DS Settings
Refresh
Properties: DS at 192.168.139.249 <V2.02f>1A
Network Properties
Outbound packets
Serial Properties
Network settings
Password
Initialize
OK
Cancel

```
Welcome to the BarrettGrasper Demo!
Winsock Started

Looking for Grasper...
I am Anakin
Broadcasting ECHO request
Waiting for UDP message...
MAC: 0.2.3.4.13.170
IP: 192.168.139.249
Port: 1001
TERMINAL MODE - Type 'X' to eXit.
_
```

Local Device Servers (auto-discovery by broadcast)
192.168.139.249 (0.2.3.4.13.170)

Routng mode: Slave
Device server IP: 192 . 168 . 139 . 249 : 1001
Subnet Mask: 255 . 255 . 255 . 0
Router IP: 127 . 0 . 0 . 1
Destination IP: 127 . 0 . 0 . 2 : 1001
Transport protocol: TCP
☐ TCP connection timeout (in minutes): 0

4 Control Modes – Supervisory and RealTime

The BarrettWraptor ™ can be used in either of two (2) modes:

1. high-level Supervisory mode or
2. low-level RealTime mode.

Most users of the BarrettWraptor ™ can rely exclusively on Supervisory mode since it handles virtually every function of the BarrettWraptor ™. Supervisory mode leverages the GOD microprocessor onboard the Hand. This processor interprets incoming Supervisory Commands and then applies control signals across the set of seven (7) motion-control microprocessors. Supervisory mode allows you to command individual or multiple motors to close, open, and move to specific positions; it also provides for setting the various configuration parameters and reporting positions and torques.

At the simplest level, Supervisory mode allows you to type and receive ASCII text characters on a terminal (using any type of computer hardware or operating system, such as UNIX, Macintosh, PalmPilot, and proprietary robot controllers, etc.). To automate grasping applications, you can write programs, scripts, or macros that send and receive these text characters through the serial port (e.g. the optional BarrettWraptor™ C-Function Library).

RealTime mode extends the capability of Supervisory mode. Sometimes users may wish to bypass the Supervisory functions and apply control directly to the motion-control microprocessors. RealTime mode enables users to close control loops in real time from their host PC or robot controller. This benefits users interested in real-time control via a data-glove, PhanTom, visual-servoing applications, real-time force control (via optional strain-gage sensors), etc. Users can switch between Supervisory and RealTime modes on-the-fly as desired, allowing for mixed mode operation.

5 Supervisory Control Mode

5.1 Overview

5.1.1 Commands

When the BarrettWraptor ™ firmware is ready to process a command, it prints a prompt of "=> " to your host computer. A command can then be entered as a single line, terminated by a carriage return character (0x0d). Once the firmware receives the carriage return, it processes the line, executes the command, prints any error result, and then prints a new prompt. Once a command has been started, no configuration changes can be made until the command has completed or has been aborted.

Many of the commands take one or more parameters; space characters should separate these from the command and each other. The command syntax is:

```
[X]<motorPrefixes><command> [<Arguments>]
```

5.1.2 Motor commands

Motor commands refer to one or more of the seven motors. By default, all seven motors are affected and the Wraptor will simulate the BarrettHand's functionality. To select fewer than seven motors, a motor prefix must be placed before the command (with no space between the prefix and the command).

Placing the optional X in front of the motor prefixes specifies that the motors listed will be exclusively acted upon. All simulation to mimic the BH-262 will be turned off, so the fingers will not move in sync, and breakaways become irrelevant. The motor prefixes and the resulting motors selected are:

Table 3 - Motor Prefixes

Value	Motor if no X is present	Motor if X is present
1	Finger 1 (motors 1 and 5)	1
2	Finger 2 (motors 2 and 6)	2
3	Finger 3 (motors 3 and 7)	3
4	Spread (motor 4)	Spread (motor 4)
5	Invalid prefix	5
6	Invalid prefix	6
7	Invalid prefix	7
G	Finger 1, Finger 2, Finger 3	Finger 1, Finger 2, Finger 3
S	Spread (motor 4)	Spread (motor 4)
<No Motor Specified>	Any motors with the parameter EN set to 1	No motor selected

Examples:

"12<command>" executes the given command on fingers 1 and 2

"X35<command>" executes the given command on motors 3 and 5

5.1.3 Status Codes

When a Supervisory mode command encounters an error or unexpected result, the command is terminated, a status code is printed, and then a new prompt is printed to the host PC. The status code is in the format "ERR <value>", where <value> is the sum of the status codes encountered. Note that the status codes are powers of 2 so that the sum may be decomposed into the individual status codes.

Table 4 - Hand Status Codes

Hand Status Code	*Description*
1	No motor board found
2	No motor found
4	Motor not initialized
8	(not used)
16	Couldn't reach position
32	Unknown command
64	Unknown parameter name
128	Invalid value
256	Tried to write a read only parameter
512	(not used)
1024	Too many arguments for this command
2048	Invalid RealTime control block header
4096	Command can't have motor prefix
8192	Overtemperature fault tripped
16384	Cntl-C abort command received

5.2 Command List

Supervisory mode commands are organized into the following siz (6) categories:

- Movement
- Motor parameter
- Global parameter
- Sequence and Macro
- Administrative
- Advanced

5.2.1 Movement Commands

Movement commands are motor commands: they immediately affect one or more of the motors. Each can take motor prefixes.

Command:	**C**
Name:	Close
Purpose:	Commands the selected motor(s) to move fingers in close direction with a velocity ramp-down at target limit. If selected, each finger moves towards the palm, and the spread motor moves so that fingers F1 and F2 are adjacent to finger F3.
Arguments:	(none)
Example:	SC
Notes:	The C command is similar to a MOVE command, with the value of the Close Target (CT) motor parameter as the destination. However, unlike the Move command, the C command will not return an ERR16 if it does not reach CT within the maximum position error (MPE) parameter.

Command:	**HI**
Name:	Hand Initialize
Purpose:	Initializes the selected motor controller(s), preparing them for use by other movement commands.
Arguments:	(none)
Example:	HI
Notes:	HI must be run before any other movement command. Generally it is run without a motor prefix, initializing all four motors; although, if desired, a subset of the motors can be specified. After an HI, all motors are in their home position; at 0 encoder counts.

Command:	**HOME**
Name:	Home
Purpose:	Moves the selected motor(s) to position 0.
Arguments:	(none)
Example:	SGHOME
Notes:	

Command:	**IO**
Name:	Incremental Open
Purpose:	Opens the selected motor(s) the given number of counts. If no is specified argument, then opens by the value of parameter DS.
Arguments:	Distance to move (0 to 20000) (optional)
Example:	12IO 5000
Notes:	

Command:	**IC**
Name:	Incremental Close
Purpose:	Closes the selected motor(s) the given number of counts. If no is specified argument, then the affected motor(s) closes by the value(s) of parameter DS.
Arguments:	Distance to move (0 to 20000) (optional)
Example:	GIC 5000
Notes:	

Command:	**LOOP**
Name:	Loop
Purpose:	Enters RealTime mode.
Arguments:	(none)
Example:	LOOP
Notes:	See Section 6 for more information on RealTime mode.

Command:	**M**
Name:	Move
Purpose:	Moves the selected motor(s) to the given position. If no argument specified, then the motor(s) move(s) to the position given by parameter DP.
Arguments:	Position (0 to 20000) (optional)
Example:	13M 1000
Notes:	

Command:	**MDEG**
Name:	Move to Degree
Purpose:	Moves the selected motor(s) to the given position expressed in degrees. If no is argument specified, then the motor(s) move(s) to the position given by DP.
Arguments:	Position in degrees (optional and can be negative)
Example:	13MDEG 90
Notes:	A position of 0º is flat on a finger motor and completely open for the spread. Larger degrees signify a closing motor, smaller degrees signify an opening motor.

Command:	**MPER**
Name:	Move to Percent
Purpose:	Moves the selected motor(s) to the given position expressed in percent of CT. If no argument is specified, then the motor(s) move(s) to the position given by DP.
Arguments:	Position in percent (optional and can be negative)
Example:	13MPER 50
Notes:	A position of 0% is flat on a finger motor and completely open for the spread. A position of 100% is at the position given by CT for all motors.

Command:	**O**
Name:	Open
Purpose:	Commands the selected motor(s) to move fingers in open direction with a velocity ramp-down at target limits. If selected, F1, F2, and F3 open away from the palm, and the spread motor moves so that fingers F1 and F2 are opposite finger F3.
Arguments:	(none)
Example:	GO
Notes:	The O command is similar to a MOVE command, with the value of the OT motor parameter as the destination. However, unlike the Move command, the O command will not return an ERR32 if it does not reach OT within the maximum position error (MPE) parameter. This command will reset the breakaway detected flag, BD.

Command:	**PTM**
Name:	Parallel Tip Move
Purpose:	Moves the selected tips in synchronization by an incremental amount so that if they are parallel, they stay parallel for the duration of the move. If no argument is specified, then the affected motor(s) move by the value of DS.
Arguments:	Distance to move (-20000 to 20000) (optional)
Example:	X567PTM -5000
Notes:	A positive increment will close motor 7 and a negative increment will open it. If the spread is open, the other tips will move the opposite direction as motor 7. If the spread is closed, the other tips will move the same direction as motor 7.

Command:	**T**
Name:	Terminate power
Purpose:	Turns the selected motor(s)'s power off.
Arguments:	(none)
Example:	ST
Notes:	Although T will return an "ERR 4" if any of the selected motor(s) isn't initialized; it will still turn the selected and initialized motor(s)'s power off.

Command:	**TC**
Name:	Torque-Controlled Close
Purpose:	Commands velocity of selected motor(s) in the direction that closes the finger(s) with control of motor torque at stall.
Arguments:	(none)
Example:	STC
Notes:	

Command:	**TO**
Name:	Torque-Controlled Open
Purpose:	Commands velocity of selected motor(s) in the direction that opens the finger(s) with control of motor torque at stall.
Arguments:	(none)
Example:	STO
Notes:	

Command:	**WM**
Name:	Wiper Move
Purpose:	Moves the selected finger(s) such that the tips remain in the same orientation with respect to the observer. If no argument specified, then the finger(s) move to the position given by DP.
Arguments:	Position (optional)
Example:	12WM 12000
Notes:	The position entered is the position that the link in the finger will finish at.

5.2.2 Motor parameter commands

Motor parameter commands act on the configuration parameters for one or more of the motors. All except FLIST can take motor prefixes. See section 5.3 for a complete list of motor parameters.

Command:	**FSET**
Name:	Finger Set
Purpose:	Sets the given parameter to the given value for the selected motor(s)
Arguments:	<parameterName> <parameterValue>
Example:	SFSET DS 100
Notes:	

Command:	**FGET**
Name:	Finger Get
Purpose:	Gets and prints the given parameter's value for the selected motor(s). Each parameter prints one value for each selected motor separated by spaces.
Arguments:	<parameterName>
Example:	SFGET DS
Notes:	

Command:	**FLOAD**
Name:	Finger Load
Purpose:	Loads the selected motor(s)'s parameters from non-volatile storage. This is done whenever the firmware starts up.
Arguments:	(none)
Example:	3FLOAD
Notes:	The non-volatile storage used does not depend on the super capacitor, so it retains its value even if the firmware is lost.

Command:	**FSAVE**
Name:	Finger Save
Purpose:	Saves the selected motor(s)'s parameters to non-volatile storage.
Arguments:	(none)
Example:	123FSAVE
Notes:	The non-volatile storage used does not depend on the super capacitor, so it retains its value even if the firmware is lost. However, this command should not be performed more than 5,000 times or the Hand electronics may need repair.

Command:	**FDEF**
Name:	Finger Default
Purpose:	Sets the selected motor(s)'s parameters back to their factory default values.
Arguments:	(none)
Example:	SFDEF
Notes:	Does not save the changed values to non-volatile storage.

Command:	**FLIST**
Name:	Finger List
Purpose:	Lists all of the standard motor parameters and their descriptions.
Arguments:	(none)
Example:	FLIST
Notes:	Does not take a motor prefix.

Command:	**FLISTV**
Name:	Finger List Value
Purpose:	Lists the motor-parameter values for the selected motor(s). Each parameter has its value(s) printed on one line, with one value for each selected motor separated by spaces.
Arguments:	(none)
Example:	3FLISTV
Notes:	

5.2.3 Global parameter commands

Global parameter commands configure the hand as a whole, without referencing a particular finger or motor. Except for not taking a motor prefix they are identical to the set of motor parameter commands. See section 5.4 for a complete list of global parameters.

Command:	**PSET**
Name:	Parameter Set
Purpose:	Sets the given parameter to the given value
Arguments:	<parameterName> <parameterValue>
Example:	PSET OTEMP 60
Notes:	

Command:	**PGET**
Name:	Parameter Get
Purpose:	Gets and prints the given parameter's value.
Arguments:	<parameterName>
Example:	PGET TEMP
Notes:	

Command:	**PLOAD**
Name:	Parameter Load
Purpose:	Loads the global parameters from non-volatile storage. This is done whenever the firmware starts up.
Arguments:	(none)
Example:	PLOAD
Notes:	The non-volatile storage used does not depend on the super capacitor, so it retains its value even if the firmware is lost.

Command:	**PSAVE**
Name:	Parameter Save
Purpose:	Saves the global parameters to non-volatile storage.
Arguments:	(none)
Example:	PSAVE
Notes:	The non-volatile storage used does not depend on the super capacitor, so it retains its value even if the firmware is lost. However, this command should not be performed more than 5,000 times or the Hand electronics may need repair.

Command:	**PDEF**
Name:	Parameter Default
Purpose:	Sets the writable global parameters to their default values.
Arguments:	(none)
Example:	PDEF
Notes:	Does not save the changed values to non-volatile storage.

Command:	**PLIST**
Name:	Parameter List
Purpose:	Lists all of the standard global parameters and their descriptions.
Arguments:	(none)
Example:	PLIST
Notes:	

Command:	**PLISTV**
Name:	Parameter List Value
Purpose:	Lists all of the standard global parameters' values. Each parameter has its value(s) printed on one line.
Arguments:	(none)
Example:	PLISTV
Notes:	

5.2.4 Sequence and macro commands

Sequence and macro commands manipulate the playing and recording of sequences and macros. Sequences and macros are automatically stored in non-volatile memory so they will survive power downs and firmware updates. A macro stores/recalls the position of every motor. The system holds a maximum of 20 macros. A sequence consists of a series of supervisory commands that are played/recorded. The system holds a maximum of 6 sequences, each of which is a maximum of 150 characters including spaces and carriage returns. Embedded sequences are not allowed, so if you are recording sequence 3, a call to play sequence 1 will not be recorded. Only the MAC command can take motor prefixes

Command:	**MAC**
Name:	Macro
Purpose:	Moves the selected motor(s) to the position specified in the macro.
Arguments:	Macro number (0-19)
Example:	GMAC 3
Notes:	The system only holds 20 macros. This command can take motor prefixes.

Command:	**MACSET**
Name:	Macro set
Purpose:	Saves the current position of the motors as a macro.
Arguments:	Macro number (0-19)
Example:	MACSET 3
Notes:	The system only holds 20 macros.

Command:	**REC**
Name:	Record
Purpose:	Start/Stop the recording of a sequence.
Arguments:	Sequence number (if starting) (0-5)
Example:	REC
Notes:	The System only holds 6 sequences. Embedded sequences are not allowed. When recording, each command sent to the hand is recorded in the sequence to be replayed later. If the sequence does not have enough free space left to record a command, the recording is stopped. If a sequence previously existed in a slot, it is erased as soon as recording starts in that slot.

Command:	**PLAY**
Name:	Play
Purpose:	Plays a previously recorded sequence
Arguments:	Sequence number (0-5)
Example:	PLAY
Notes:	The system only holds 6 sequences. Embedded sequences are not allowed.

Command:	**WAIT**
Name:	Wait
Purpose:	Causes the system to wait a given number of milliseconds
Arguments:	Milliseconds to wait
Example:	WAIT 5000
Notes:	This command is useful in sequences.

5.2.5 Administrative commands

Administrative commands implement various housekeeping functions.

Command:	**?**
Name:	Help
Purpose:	Lists all of the standard commands. If immediately followed by a command name, then it lists the command name and its description.
Arguments:	<commandName>
Example:	?HI
Notes:	There must be no space between "?" and any command name.

Command:	**RESET**
Name:	Reset
Purpose:	Resets the hand software. Equivalent to doing a power cycle.
Arguments:	(none)
Example:	RESET
Notes:	

Command:	**ERR**
Name:	Error
Purpose:	If given an argument, lists the errors represented by that argument. If not given an argument, lists all possible error values and descriptions.
Arguments:	<errorNum>
Example:	ERR 3
Notes:	

Command:	**VERS**
Name:	Version
Purpose:	Prints the firmware version.
Arguments:	(none)
Example:	VERS
Notes:	

5.2.6 Advanced commands

Users do not generally need these commands and should avoid using them. They are not listed by the “?" command; they are only listed by the "A?" command.

Command:	**A?**
Name:	Help All
Purpose:	Lists all of the standard and advanced commands.
Arguments:	(none)
Example:	A?
Notes:	Use the ? command to get a description of an advanced command.

Command:	**FLISTA**
Name:	Finger List All
Purpose:	Lists all of the standard and advanced finger parameter names.
Arguments:	(none)
Example:	FLISTA
Notes:	

Command:	**FLISTAV**
Name:	Finger List All Value
Purpose:	Lists all of the standard and advanced motor parameter's values for the selected motor(s). Each parameter has its value(s) printed on one line, with one value for each selected motor separated by spaces.
Arguments:	(none)
Example:	3FLISTAV
Notes:	Can take a motor prefix.

Command:	**PLISTA**
Name:	Parameter List All
Purpose:	Lists all of the standard and advanced global parameter names.
Arguments:	(none)
Example:	PLISTA
Notes:	

Command:	**PLISTAV**
Name:	Parameter List All Value
Purpose:	Lists all of the standard and advanced global parameter's values. Each parameter has its value printed on one line.
Arguments:	(none)
Example:	PLISTAV
Notes:	

5.3 Motor Parameters

Motor parameters change how a motor functions.

5.3.1 Movement

Movement parameters affect how a given motor moves.

Parameter:	**BS**
Name:	Breakaway Stop
Purpose:	Used to stop finger as soon as breakaway has been detected.
Values:	0 or 1
Default:	Link: 0 Spread: N/A Tip: 0
Notes:	When set to 1, the finger’s motors will stop when breakaway is detected. Breakaways only occur when BarrettHand functionality is being simulated (i.e. no “X” motor prefix).

Parameter:	**DP**
Name:	Default Position
Purpose:	Destination of M command if no argument specified
Values:	-32,767 to 32,767
Default:	Link: 8500 Spread: 1575 Tip: 8500
Notes:	While DP has a range from -32,767 to 32,767, its true range of useful values is bounded by the joint limits of the axes (e.g. approx. -1400 to 18,500 for link, approx. 0 to 9000 for spread and approx. –8700 to 19000 for tip).

Parameter:	**DS**
Name:	Default Step
Purpose:	Size of IC or IO command movement if no argument specified
Values:	-32,767 to 32,767
Default:	Link: 1400 Spread: 800 Tip: 1400
Notes:	While DS has a range from -32,767 to 32,767, its true range of useful values is bounded by the joint limits of the axes (e.g. approx. -1400 to 18,500 for link, approx. 0 to 9000 for spread and approx. –8700 to 19000 for tip).

Parameter:	**HSG**
Name:	Highest Strain Gauge Value
Purpose:	In O, C, IO, IC, M, TO, TC, and HOME commands, a motor's motion is terminated if its strain gauge value exceeds HSG.
Values:	0 to 256
Default:	256
Notes:	This command is an alias for MSG. When writing new code, it is recommended to use this command instead of MSG. A value of 255 or 256 disables the strain gage checking during motion commands.

Parameter:	**LSG**
Name:	Lowest Strain Gauge Value
Purpose:	In O, C, IO, IC, M, TO, TC, and HOME commands, a motor's motion is terminated if its strain gauge value falls below LSG.
Values:	0 to 256
Default:	256
Notes:	A value of 255 or 256 disables the strain gage checking during motion commands.

Parameter:	**MOV**
Name:	Maximum Open Velocity
Purpose:	Controls the maximum velocity while opening a motor.
Values:	16 to 4080
Default:	1000
Notes:	

Parameter:	**MCV**
Name:	Maximum Close Velocity
Purpose:	Controls the maximum velocity while closing a motor.
Values:	16 to 4080
Default:	1000
Notes:	

Parameter:	**MSG**
Name:	Maximum Strain Gauge
Purpose:	In O, C, IO, IC, M, TO, TC, and HOME commands, a motor's motion is terminated if its strain gauge value exceeds MSG.
Values:	0 to 256
Default:	256
Notes:	HSG is the preferred alias to MSG. When possible, please use HSG in place of MSG. A value of 255 or 256 disables the strain gage checking during motion commands.

5.3.2 Status

Motor status parameters are read-only and give information about the state of a motor.

Parameter:	**BD**
Name:	Breakaway-Detected Flag
Purpose:	To determine if breakaway has occurred
Values:	0 or 1
Default:	N/A
Notes:	Flag is set when a breakaway is detected.

Parameter:	**BP**
Name:	Breakaway Position
Purpose:	Stores position of last breakaway
Values:	0 or 20000
Default:	N/A
Notes:	Stored position of location of last breakaway.

Parameter:	**OD**
Name:	Odometer
Purpose:	The total number of counts traveled by the selected motor, divided by 1000.
Values:	0 to 4 billion
Default:	N/A
Notes:	This value is never reset; it is maintained through power failures and firmware downloads.

Parameter:	**P**
Name:	Position
Purpose:	The present position of the motor.
Values:	Link: -1400 to approximately 18500 Spread: 0 to approximately 9100 Tip: -8700 to approximately 19000
Default:	N/A
Notes:	

Parameter:	**S**
Name:	Status
Purpose:	The present status of the motor. 0 if ready to be used, or a status code otherwise.
Values:	0, 1, 2, 4,…8192, 16384 and sums of these
Default:	N/A
Notes:	See status codes in section 5.1.3.

Parameter:	**SG**
Name:	Strain Gauge
Purpose:	The present strain gage value for the motor.
Values:	0 to 255
Default:	N/A
Notes:	

5.3.3 RealTime

RealTime parameters affect the control and feedback data for a motor while in RealTime mode. See Section 6 for more information on RealTime mode.

Parameter:	**LCV**
Name:	Loop Control Velocity
Purpose:	If non-zero, then a velocity byte will be sent in the control block for the motor.
Values:	0, 1
Default:	1
Notes:	

Parameter:	**LCVC**
Name:	Loop Control Velocity Coefficient
Purpose:	When the firmware receives a velocity byte in a control block, it multiplies the value by the value of LCVC before passing it to the affected motor.
Values:	0 to 255
Default:	8
Notes:	

Parameter:	**LCPG**
Name:	Loop Control Proportional Gain
Purpose:	If non-zero, then a Proportional Gain byte will be sent in the control block for the motor.
Values:	0, 1
Default:	1
Notes:	This controls the constant that is multiplied by the velocity error in order to produce the motor torque.

Parameter:	**LFV**
Name:	Loop Feedback Velocity
Purpose:	If non-zero, then the firmware sends a signed byte giving the present velocity for the motor divided by the LFVC parameter.
Values:	0, 1
Default:	1
Notes:	

Parameter:	**LFVC**
Name:	Loop Feedback Velocity Coefficient
Purpose:	Before sending a Loop Feedback Velocity byte, the firmware divides the velocity by this parameter's value.
Values:	0 to 255
Default:	8
Notes:	

Parameter:	**LFS**
Name:	Loop Feedback Strain
Purpose:	If non-zero, then the firmware sends an unsigned byte giving the strain gauge value for the motor.
Values:	0, 1
Default:	1
Notes:	

Parameter:	**LFAP**
Name:	Loop Feedback Absolute Position
Purpose:	If non-zero, then the firmware sends an unsigned two-byte value giving the present position of the motor.
Values:	0, 1
Default:	1
Notes:	

Parameter:	**LFDP**
Name:	Loop Feedback Delta Position
Purpose:	If non-zero, then the firmware sends a signed byte giving the change in position since the last datum, divided by the value of the LFDPC parameter.
Values:	0, 1
Default:	1
Notes:	A conflict occurs when the change in position is too great to transmit in a single signed byte, even after scaling. See section 6.2.3 for more information.

Parameter:	**LFDPC**
Name:	Loop Feedback Delta Position Coefficient
Purpose:	Used to scale a delta position value.
Values:	0 to 255
Default:	1
Notes:	Delta position is the change in position from the last reported position and is limited to one signed byte. The Present position is read and compared to the last reported position. The difference is divided by the RealTime variable LFDPC, clipped to a single signed byte, and then sent to the host. The value sent to the host should be multiplied by LFDPC and then added to the last reported position

5.3.4 Advanced

Users do not generally need these commands and should avoid using them. They are not listed by the FLIST or FLISTV commands; they are only listed by the FLISTA and FLISTAV commands.

Parameter:	**ACCEL**
Name:	Acceleration
Purpose:	Maximum acceleration and deceleration when moving from one position to another.
Values:	0 to 65,535
Default:	Link: 40
	Spread: 30
	Tip: 30
Notes:	While the ACCEL parameter has a rather large range of values that it can accept, the motor can only follow a small subset of those values. In general, the useful range is from 0 to approx. 60. Above 60, the motors cannot provide enough torque to accelerate that quickly. The units for acceleration is defined as 1024*sqrt(Hall count)/ms

Parameter:	**CT**
Name:	Close Target
Purpose:	This is the position gone to by a C ("Close") command.
Values:	-32,767 to 32,767
Default:	Link: 18,200
	Spread: 9300
	Tip: 18,200
Notes:	While CT has a range from -32,767 to 32,767, its true range of useful values is bounded by the joint limits of the axes (e.g. approx. -1400 to 18,500 for link, approx. 0 to 9000 for spread and approx. –8700 to 19000 for tip).

Parameter:	**EN**
Name:	Enabled
Purpose:	If non-zero, then a motion command with no motor prefix will act on this motor.
Values:	0, 1
Default:	1
Notes:	

Parameter:	**HOLD**
Name:	Hold
Purpose:	If non-zero, then the motor is left energized after each motion command in order to hold the position constant.
Values:	0, 1
Default:	Link: 0
	Spread: 1
	Tip: 0
Notes:	Since the fingers are not back-drivable, this is generally set to 1 only for the spread motor.

Parameter:	**MPE**
Name:	Maximum Position Error
Purpose:	After moving to a desired position, if the position error is less than MPE then the move is considered a success.
Values:	-32,767 to 32,767
Default:	200
Notes:	While MPE has a range from -32,767 to 32,767, its true range of useful values is bounded by the joint limits of the axes (e.g. approx. -1400 to 18,500 for link, approx. 0 to 9000 for spread and approx. –8700 to 19000 for tip).

Parameter:	**OT**
Name:	Open Target
Purpose:	This is the position gone to by an O ("Open") command.
Values:	-32,767 to 32,767
Default:	0
Notes:	While OT has a range from -32,767 to 32,767, its true range of useful values is bounded by the joint limits of the axes (e.g. approx. -1400 to 18,500 for link, approx. 0 to 9000 for spread and approx. –8700 to 19000 for tip).

Parameter:	**SAMPLE**
Name:	Sample Time
Purpose:	Controls the sample frequency of the motor controller chip.
Values:	31
Default:	31
Notes:	Read only value.

Parameter:	**TSTOP**
Name:	Time to Stop
Purpose:	Time in milliseconds before motor is considered stopped.
Values:	0 to 65,535
Default:	Link: 300 Spread: 400 Tip: 300

Notes: WARNING: Please use caution when adjusting this parameter. Setting TSTOP higher than its default can result in the motors heating up very quickly under moderate to heavy usage.

5.4 Global Parameters

Global parameters are used to configure or observe the hand as a whole.

5.4.1 Configuration

Global configuration parameters affect the hand as a whole.

Parameter:	**BAUD**
Name:	Baud rate
Purpose:	Controls the serial port baud rate. Value is baud rate divided by 100.
Values:	6, 12, 24, 48, 96, 192, 384.
Default:	96
Notes:	

Parameter:	**LFT**
Name:	Loop Feedback Temperature
Purpose:	If non-zero, then when in RealTime mode the firmware sends a signed two-byte datum of temperature in each feedback block.
Values:	0, 1
Default:	0
Notes:	

Parameter:	**OTEMP**
Name:	OverTemperature
Purpose:	If not -551, then if the temperature exceeds this value then any motor command fails with an overtemperature error.
Values:	-551 to 1250
Default:	-551
Notes:	Value is temperature in tenths of a degrees C.

5.4.2 Status

Global status parameters are read-only and give information about the state of the hand.

Parameter:	**TEMP**
Name:	Temperature
Purpose:	The present temperature on the CPU board in tenths of a degree C.
Values:	-550 to 1250
Default:	N/A
Notes:	

Parameter:	**PTEMP**
Name:	Peak Temperature
Purpose:	The maximum temperature ever experienced by this hand
Values:	0 to 1250
Default:	N/A
Notes:	This value is never reset; it is maintained through power failures and firmware downloads.

Parameter:	**UPSECS**
Name:	Uptime Seconds
Purpose:	The total power-up time for this hand.
Values:	0 to 4 billion
Default:	N/A
Notes:	This value is never reset; it is maintained through power failures and firmware downloads. The parameter can accommodate 136 years of power-up time before rolling over.

Parameter:	**SN**
Name:	Serial Number
Purpose:	The serial number of the hand.
Values:	N/A
Default:	N/A
Notes:	This value is never reset; it is maintained through power failures and firmware downloads.

5.4.3 Advanced

Parameter:	**LFDPD**
Name:	Loop Feedback Delta Position Discard
Purpose:	If non-zero, then in RealTime mode any position change that cannot be sent in a delta position datum is discarded. If zero, then any unsent position change is accumulated for transmission in the next cycle.
Values:	0, 1
Default:	0
Notes:	

5.5 Termination Conditions for Movement Commands

There are twelve commands in Supervisory mode that control finger motion:

• Position commands, absolute:	M, MDEG, MPER, HOME, WM
• Position commands, relative:	IO, IC, PTM.
• Velocity commands with ramp-down at target limits:	O, C
• Velocity commands with control of motor torque at stall:	TO, TC

In all cases, the command terminates and returns when, for every motor specified in the command, one of the following termination conditions applies:

Case 1: Motor stalls because obstacle(s) stop the motion. The obstacles include foreign objects as well as joint stops and other fingers.

- When a motor stalls, the controller will continue driving it for TSTOP milliseconds, after which a termination condition occurs.
- If HOLD is false, the motor is then turned off; if HOLD is true, the motor is servoed to maintain this position.
- For position commands, if the termination position is not within MPE (Maximum Position Error), the status code ERR16 is returned corresponding to “Couldn’t reach position.”

Case 2: Specified goal position is achieved within MPE.

- As soon as the goal position is reached, a termination condition occurs for that motor.
- If HOLD is false, the motor is then turned off; if HOLD is true, the motor is servoed to maintain the position at which the termination condition occurred.
- No status code is returned, and a query to parameter S returns 0.

Case 3: Control-C Character Sent

- As soon as the Control C is received, a termination condition occurs for all motors.
- For each motor, if HOLD is false, that motor is then turned off; if HOLD is true, that motor is servoed to maintain the position at which the termination condition occurred.
- The status code returned is 16384.

6 RealTime Control

6.1 Overview

RealTime mode, also known as Loop-Control mode, is the second control method for the BarrettWraptor. This control mode allows you to send control data continuously and receive feedback data, without waiting for the motors to stop moving. Any desired control law can be implemented within the host computer by calculating the desired motor command, sending that command to the BarrettWraptor, waiting for the requested feedback data, and then calculating the next motor command. The control bandwidth is a function of the amount of control data sent, the amount of feedback data requested and the chosen baud rate.

Control data from the host computer to the hand is grouped into control blocks; feedback data from the BarrettWraptor ™ is grouped into feedback blocks. The structure of the control and feedback blocks is set by various finger and global parameters. The structure can only be changed in Supervisory mode; it cannot be changed while in RealTime mode.

While in RealTime mode, the firmware keeps track of the present and desired velocities. It calculates the motor torque by multiplying the velocity error by the proportional gain value.

To enter RealTime mode, the host computer sends the Supervisory mode "LOOP" command, specifying the motors to be controlled. The Hand responds with an acknowledgment character ("*"), and then awaits control blocks, with or without control data. When a control block is received, if the control block requests a feedback block, then transmission of the feedback block is started. Once the complete control block is received it is acted upon, and then the hand waits for the next control block. The host should not send a second control block until the first one is acknowledged.

If the Hand software encounters an error, then the next time the Hand would send an “*” character to the host it instead sends "<CRLF>ERR" followed by the error value. It then returns to Supervisory mode.

To terminate RealTime mode, the host should send a single ^C character instead of the header character. This returns the Hand to Supervisory mode.

6.2 Control and Feedback Blocks

Control and feedback blocks consist of a header character, followed if desired by control data. If control data is included then it is sent for each motor selected for the LOOP command, in motor number order, followed by any global datum. For each motor, any of a set of data can be included. Whether or not a specific piece of data should be included is controlled by one of seven flag parameters: "LCV", "LCPG", "LFV", "LFS", "LFAP", "LFDP" and "LFT." If a given parameter is true then its corresponding datum is included in the block; if not, then it is omitted. (Three other parameters, "LFVC", "LFDPC", and "LFDPD", modify specific data items.)

6.2.1 Control Blocks

Control data from the host to the hand is grouped into control blocks. Each control block has a single byte header, optionally followed by a set of control data. The header specifies whether or not control data is to follow, and whether or not a feedback block is to be returned. The header can also terminate RealTime mode.

The possible header byte values are:

"C": Control data follows; respond with a feedback block

"c": Control data follows; respond with an acknowledgment character

"A": No control data follows; respond with a feedback block

"a": No control data follows; respond with an acknowledgment character

<^C>: Terminate RealTime mode

If the "C" or "c" header is used, then the header should be followed by control data. For each motor, two different data values should be included in order if their corresponding flags are true:

LCV ("Loop Control Velocity"): 1 signed byte

LCPG ("Loop Control Proportional Gain"): 1 unsigned byte

The control data should be sent in a specific order: first all data for motor 1, then all for motor 2, then motor 3, then motor 4, then motor 5, then motor 6 and finally motor 7. Note that if a given motor was not specified in the initiating LOOP command, or if a specific value isn't enabled by the corresponding finger parameter, then the corresponding datum should not be transmitted.

If the LCV datum is included, then the hand will multiply it by the parameter LCVC before passing it on to the motor.

6.2.2 Feedback Blocks

Data from the hand to the host is grouped into feedback blocks. Each feedback block has a single byte header ("*"), followed (if requested) by a set of feedback data. If the hand has encountered an error, then the header is replaced by "<CRLF>ERR ", followed by the error number; the hand then returns to supervisory command mode.

For each selected motor, four different data values are included in order if their corresponding finger parameters are non-zero:

LFV ("Loop Feedback Velocity"): 1 signed byte

LFS ("Loop Feedback Strain"): 1 unsigned byte

LFAP ("Loop Feedback Absolute Position"): Signed 4-byte word

LFDP ("Loop Feedback Delta Potion"): 1 signed byte

In addition to the motor feedback data, there is a single global feedback datum, which is sent if its corresponding global parameter is non-zero:

LFT ("Loop Feedback Temperature"): Signed 2-byte word

The feedback data are sent in a specific order: first all data for motor 1, then all for motor 2, then motor 3, then motor 4, then motor 5, then motor 6, then motor 7, and finally any global datum. Note that if a given motor was not specified in the initiating LOOP command, or if a specific value isn't enabled by the corresponding finger or global parameter, then the corresponding datum is not transmitted.

If the LFV datum is included, then the hand will divide it by the parameter Loop LFVC before sending it to the host.

6.2.3 Loop Feedback Delta Position

The LFDP ("Loop Feedback Delta Position") datum is a special case. Each time a motor's position is queried using "FGET P", the reported position is remembered. In loop mode, if the LFDP parameter is non-zero then the present position is read and compared to the previously reported position. The difference is divided by the LFDPC ("Loop Feedback Delta Position Coefficient") parameter, clipped to a single signed byte, and then sent to the host. The host should then multiply the received value by LFDPC and then add it to the reported position.

The problem with using delta position is that the reported position can change at most by +127/-128 in each cycle. If the motor position changes more than this in a single cycle then the reported position will lag behind the actual position.

Example: say LFDPC is 2, the last reported position was 1500, and the position suddenly jumps to 2000. The first feedback block will include the delta position datum 127, which actually means 254; the hand will internally update the reported position to 1754. The next feedback block will include the delta position 123, which actually means 246; the reported position will be updated to 2000. Subsequent feedback blocks will include the delta position value 0 (until the next position change).

If desired, any unreported position change can be discarded by setting the LFDPD ("Loop Feedback Delta Position Discarded") global parameter to true. With this set, the above example would result in the single value 127 being sent to the host, followed by 0s.

6.3 Parameter Summary

This is a summary of the different motor and global parameters, which affect RealTime mode. Most of the parameters are flags, specifying whether a specific datum is to be present in a control or feedback block. The four remaining parameters are coefficients or flags, which affect how the firmware interprets or generates a datum.

Table 5 - RealTime Finger Control Parameters

Parameter	Name	Type	Function	Size in Block
LCV	Loop Control Velocity	Flag	If True, RealTime control block will contain control velocity	1 signed byte
LCVC	Loop Control Velocity Coefficient	Coefficient (1 to 255)	LCV is multiplied by LCVC to determine control velocity	N/A
LCPG	Loop Control Proportional Gain	Flag	If True, RealTime control block will contain Proportional Gain	1 unsigned byte
LFV	Loop Feedback Velocity	Flag	If True, RealTime feedback block will contain feedback velocity	1 signed byte
LFVC	Loop Feedback Velocity Coefficient	Coefficient (1 to 255)	Actual velocity is divided by LFVC to get LFV	N/A
LFS	Loop Feedback Strain	Flag	If True, RealTime feedback block will contain strain information	1 unsigned byte
LFAP	Loop Feedback Absolute Position	Flag	If True, RealTime feedback block will contain absolute position	4 signed bytes
LFDP	Loop Feedback Delta Position	Flag	If True, RealTime feedback block will contain delta position	1 signed byte
LFDPC	Loop Feedback Delta Position Coefficient	Coefficient (1 to 255)	The actual delta position is divided by this to get LFDP	N/A
LFDPD	Loop Feedback Delta Position Discard	Flag	If true, any delta position overflow is discarded	N/A

Table 6 - RealTime Global Control Parameters

Parameter	Name	Type	Function	Size in Block
LFT	Loop Feedback Temp.	Flag	If True, RealTime feedback block will contain temperature	2 signed bytes

6.4 Example

This application uses fingers 1 and 2, and the spread. The fingers will receive velocity control information and report strain and delta position. The spread will just report delta position. The feedback block will also include the present hand temperature. All relevant coefficients will be set to 1.

To set this, use the following commands:

```
12FSET LCV 1
12FSET LCVC 1
12FSET LCPG 0
12FSET LFV 0
12FSET LFS 1
12FSET LFAP 0
12FSET LFDP 1
12FSET LFDPC 1
4FSET LCV 0
4FSET LCPG 0
4FSET LFV 0
4FSET LFS 0
4FSET LFAP 0
4FSET LFDP 1
4FSET LFDPC 1
PSET LFT 1
124LOOP
```

The hand will then send a single "*" and wait for control blocks. Each control block will consist of three bytes:

"C" [Control data follows; respond with feedback block]

1 signed byte of velocity for motor 1

1 signed byte of velocity for motor 2

Each feedback block will consist of eight bytes:

"*"

1 unsigned byte of strain for motor 1

1 signed byte of delta position for motor 1

1 unsigned byte of strain for motor 2

1 signed byte of delta position for motor 2

1 signed byte of delta position for motor 4

2-signed bytes of temperature

Each control block from the host will stimulate a feedback block from the hand. When the host is finished, it will send the single character ^C (0x03); the hand will respond by printing a prompt and waiting for a new command.

7 C Function Reference

`close`

Syntax:
```
void close(SOCKET g, const char *nodes);
```
Arguments:

g:	socket returned by tcp_connect(),
nodes:	null-terminated string of integers specifying fingers

Example:
```
/* Close fingers 1 and 3 */
close(g, "13");
```
Notes: Actually issues a move() command for the fingers specified, plus their fingertips. Automatically calculates a corresponding end position for the fingertip motor.

`command`

Syntax:
```
void command(SOCKET g, char *message);
```
Arguments:

g:	socket returned by `tcp_connect()`
message:	null-terminated command string to send to Wraptor

Example:
```
/* Set motor 7's Max Torque Limit to 550 */
command(g, "7 S MT 550");
```
Notes: If the command sets any motor to Trapezoidal Mode, the function will wait until those motors complete their movement.

`delay`

Syntax:
```
void delay(DWORD ms);
```
Arguments:

ms:	number of milliseconds to delay execution

Example:
```
/* Wait for 2.5 seconds */
delay(2500);
```
Notes: While the code is delayed, incoming messages from the Wraptor are still received and processed.

`discoverGrasper`

Syntax:
```
int discoverGrasper(char mac[], char ipaddress[], int *port);
```
Arguments:

mac:	Returned MAC address of the found Wraptor
ipaddress:	Returned IP address of the found Wraptor
port:	Returned port number of the found Wraptor

Example:
```
/* Find the Wraptor on the network */
int port;
char mac[127], ipaddress[127];
discoverGrasper(mac, ipaddress, &port);
```
Notes: The present implementation allows for only <u>one</u> Wraptor on each network. The function uses a UDP broadcast to locate the Wraptor.

`downloadFirmware`

Syntax:
```
int downloadFirmware(SOCKET g, const char *nodes, char *fname);
```
Arguments:

g:	socket returned by `tcp_connect()`
nodes:	null-terminated string of integers specifying motor controllers
fname:	pathname of the firmware (*.tek) file to be downloaded

Example:
```
/* Download new firmware */
char fname[127], nodes[16];
printf("Load Firmware: ");
gets(fname);
printf("To which nodes: ");
gets(nodes);
downloadFirmware(g, nodes, fname);
```
Notes: The function shows the percent downloaded on-screen. Barrett strongly recommends downloading to only <u>one</u> node (motor controller) at a time to avoid firmware corruption. Download time is approximately 10 minutes. The motor controllers are put into their "download firmware" state by cycling power, or by setting their STATUS property to zero.

findZero

Syntax:	`void findZero(SOCKET g, const char *nodes);`	
Arguments:	g:	socket returned by `tcp_connect()`
	nodes:	null-terminated string of integers specifying motor controllers

Example:	`/* Find the zero-point on all motors */`
	`findZero(g, “1234567”);`	

Notes:	The function first tries to open motors 1, 2, and 3 to their stops. Then it opens motor 4 (spread) while closing the fingertips (5, 6, and 7). It then resets the ActualPositions to known values and moves all joints to a position of zero. Make sure the Wraptor's workspace is clear of obstructions before calling this function.	

get

Syntax:	`void get(SOCKET g, const char *nodes, int property);`	
Arguments:	g:	socket returned by `tcp_connect()`
	nodes:	null-terminated string of integers specifying motor controllers
	property:	integer index of the property to be retrieved

Example:	`/* Get the temperature of motor controllers 5, 6, and 7 */`
	`get(g, “567”, MOTOR_TEMPERATURE);`	

Notes:	The retrieved property values are placed into the propertyList[][] array. The function waits until all motors have responded with their value.	

initSoftware

Syntax:	`void initSoftware(void);`
Arguments:	none

Example:	`/* Initialize the local Wraptor property arrays to zero */`
	`initSoftware();`

Notes:	Must be called before communicating with the Wraptor.

Listener

Syntax:	`DWORD WINAPI Listener(void *g);`	
Arguments:	g:	socket returned by `tcp_connect()`

Example:	`/* Launch the TCP Listener */`
	`DWORD nThreadID;`
	`CreateThread(0, 0, Listener, (void*)g, 0, &nThreadID);`	

Notes:	The Listener thread runs in an endless loop listening for packets sent by the Wraptor then processing them.	

move

Syntax:	`void move(SOCKET g, const char *nodes, long value);`	
Arguments:	g:	socket returned by `tcp_connect()`
	nodes:	null-terminated string of integers specifying fingers
	value:	position to move to (in hall counts)

Example:	`/* Move fingers 1 and 2 to position 250 */`
	`move(g, “12”, 250);`	

Notes:	Automatically calculates a corresponding end position for the fingertip motor.	

open

Syntax:	`void open(SOCKET g, const char *nodes);`	
Arguments:	g:	socket returned by tcp_connect(),
	nodes:	null-terminated string of integers specifying fingers

Example:	`/* Open fingers 1 and 3 */`
	`open(g, “13”);`	

Notes:	Actually issues a move() to position zero command for the fingers specified, plus their fingertips. Automatically calculates a corresponding end position for the fingertip motor.	

saveParameters

Syntax:	`void saveParameters(SOCKET g, const char *nodes);`
Arguments:	g: socket returned by tcp_connect(),
	nodes: null-terminated string of integers specifying motor controllers (or GOD)
Example:	`/* Save the Wraptor parameters, prepare for power-off */`
	`saveParameters(g, "1234567");`
Notes:	Should be issued while each motor control is in Control Mode MODE_IDLE, before removing power from the Wraptor, in order to save the present motor positions and control parameters.

set

Syntax:	`void set(SOCKET g, const char *nodes, int property, long value);`
Arguments:	g: socket returned by `tcp_connect()`
	nodes: null-terminated string of integers specifying motor controllers
	property: integer index of the property to be set
	value: value to set the property to
Example:	`/* Set the trapezoidal end position of motors 5, 6, and 7 to -85 */`
	`set(g, "567", TRAP_END_POSITION, -85);`
Notes:	none

tcp_connect

Syntax: `SOCKET tcp_connect(const char *hostname, unsigned short portnum);`

Arguments: hostname: hostname or IP address of Wraptor
portnum: port number for communication (typically 1001)

Example:

```
/* Establish a TCP socket connection to the Wraptor */
if((g = tcp_connect(ipaddress, port)) == -1){
       printf("\nError: Can not connect to Wraptor!");
       exit(-1);
}
```

Notes: See the GrasperTest main.c example program.

tcp_msg

Syntax:	`SOCKET tcp_msg(SOCKET sd, const char *buff, int len);`
Arguments:	sd: socket returned by `tcp_connect()`
	buff: (unterminated) string buffer of characters to send to Wraptor
	len: number of characters to send from buff
Example:	`/* Send a character string to the Wraptor */`
	`tcp_msg(g, "1 G AP", 6);`
Notes:	Issues a raw stream of characters to the Wraptor.

waitFor

Syntax:	`void waitFor(SOCKET g, const char *nodes, int property, long value);`
Arguments:	g: socket returned by `tcp_connect()`
	nodes: null-terminated string of integers specifying motor controllers
	property: integer index of the property to be set
	value: value to set the property to
Example:	`/* Wait for the Control Mode to be MODE_PID */`
	`waitFor(g, "567", CONTROL_MODE, MODE_PID);`
Notes:	Waits in a loop until the propertyList[][] entry for all specified motors matches the given value.

`wakeGrasper`

Syntax:	`void wakeGrasper(SOCKET g);`
Arguments:	g: socket returned by `tcp_connect()`

Example:	`/* Prepare the Wraptor for initial commands */`
	`wakeGrasper(g);`

Notes:	When the Wraptor is first powered-up (or when STATUS is set to zero), all motor controllers (and GOD) are in a "Wait for firmware update" state- they will not respond to motion commands properly. Calling this function gets the motor controllers into a "Ready" state. NOTE: If you are updating firmware, you must <u>not</u> call this command before `downloadFirmware()`, else the download will fail.

`ws_down`

Syntax:	`int ws_down();`
Arguments:	none

Example:	`/* Shut down WinSock, break communication with Wraptor */`
	`ws_down();`

Notes:	Call this at the end of your program.

`ws_init`

Syntax:	`int ws_init();`
Arguments:	none

Example:	`/* Start up WinSock */`
	`ws_init();`

Notes:	Call this at the beginning of your program.

8 Kinematics

Homogeneous Transform Between {*i-1*} and {*i*}
D-H Parameter Values for all Fingers
Forward Kinematics from Fingertip to World
D-H Frame Assignment (**image**)
D-H Link Parameters for Fingers
Forward Kinematics for Finger
Motor to Joint Angle Transform

9 Maintenance

The BarrettWraptor is designed to be maintenance-free.

10 Troubleshooting

Symptom / Possible solution
This section will be expanded as common problems are determined.

11 FAQ

This section will be expanded as questions arise.